



Parle Tilak Vidyalaya Associations

SATHAYE COLLEGE (Autonomous)

Vile-Parle (East), Mumbai – 400 057.

Practical Journal

Blockchain

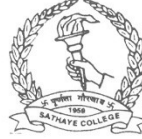
Submitted by

Vaishnavi Kasare

Seat No.: **15**

M.Sc. [I.T.]-Information Technology Part II

2022 – 2023



**Parle Tilak Vidyalaya Association's
SATHAYE COLLEGE (Autonomous)**

Vile-Parle (East), Mumbai – 400 057.

CERTIFICATE

This is to certify that Vaishnavi Vishnu Kasare

Seat No 15 has successfully completed all the practicals in
the subject of Blockchain for M.Sc.I.T. Part-II SEM – IV as
prescribed by University of Mumbai for the year 2022-2023.

Coordinator	Professor in Charge	External Examiner
M.Sc. [I.T.]		

Date:	Date	Date
-------	------	------

Sathaye College

Vaishnavi Kasare

INDEX

Sr.No	Date	Practical Title	Sign
1	11.02.2023	Practical No 1 Create blockchain with 3 blocks and hence display the entire blockchain, hash value and timestamp of each block.	
2	17.02.2023	Practical No 2 a) Create a smart contract for counter. b) Create a smart contract for calculator.	
3	17.03.2023	Practical No 3 a) Write a solidity program to demonstrate array and its types. b) Demonstrate the use of loops in solidity.	
4	24.03.2023	Practical No 4 a) Solidity program to demonstrate Comparison operators. b) Solidity program to demonstrate Logical operators. c) Solidity program to demonstrate Assignment operators. d) Solidity program to demonstrate Ternary operators. e) Solidity program to demonstrate Bitwise operators.	
5	31.03.2023	Practical No 5 Write a solidity program to find if the number is odd or even, prime or composite.	

6	31.04.2023	Practical No 6 Write a solidity program to find modulus of addition of two numbers with DD and modulus of multiplication of two numbers with MM where the two numbers are achieved by performing AND operation of DD and MM of your date of birth and OR operation of YY and YY from year of birth.	
7	04.05.2023	Practical No 7 a) Create a smart contract to demonstrate Mathematical function. b) Create a smart contract to demonstrate Function overloading.	
8	08.05.2023	Practical No 8 a) Create a smart contract to show the implementation of Interface. b) Create a smart contract to show the implementation of Inheritance.	
9	10.05.2023	Practical No 9 Write a solidity program to create an array of roll numbers and then create a smart contract where it checks the value of the roll number and perform AND operation with today's date and if the result is even then allow the student else deny (DD part only).	
10	11.05.2023	Practical No 10 Write a solidity program to find the sum of an array of ten numbers using loop the numbers are expected to be taken from the user, create a smart contract to find the AND operation of odd positioned numbers and OR operation of even positioned numbers including 0 th index. Hence find the product of the results and also identify whether the result is the part of array or not.	

Practical No 1

Aim: Create blockchain with 3 blocks and hence display the entire blockchain, hash value and timestamp of each block.

Code:

File Edit Format Run Options Window Help

```
import datetime
import hashlib

class Block:
    def __init__(self, previous_block_hash, data, timestamp):
        self.previous_block_hash = previous_block_hash
        self.data = data
        self.timestamp = timestamp
        self.hash = self.get_hash()

    @staticmethod
    def create_genesis_block():
        return Block("0", "0", datetime.datetime.now())

    def get_hash(self):
        header = (str(self.previous_block_hash) + str(self.data) + str(self.timestamp))
        inner_hash = hashlib.sha256(header.encode()).hexdigest().encode()
        comp_hash = hashlib.sha256(inner_hash).hexdigest()
        return comp_hash

number_of_blocks = 3

Blockchain = [Block.create_genesis_block()]
print("Genesis block is created")
print("Hash: %s" % Blockchain[0].hash)

for i in range(1, number_of_blocks):
    Blockchain.append(Block(Blockchain[i-1].hash, "Block number %d" % i, datetime.datetime.now()))
    print("%d block created" % i)
    print("Hash: %s" % Blockchain[-1].hash)
    print("timestamp: ", datetime.datetime.now())
```

Output:

IDLE Shell 3.10.0

File Edit Shell Debug Options Window Help

```
Python 3.10.0 (tags/v3.10.0:b494f59, Oct 4 2021, 19:00:18) [MSC v.1929
AMD64] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
= RESTART: C:/Users/mazhar/AppData/Local/Programs/Python/Python310/block
va quesion.py
Genesis block is created
Hash: 80069ac29e94bdbfd44d7dcef3be07dae116e7be746171439000fa308335637
1 block created
Hash: 48d63604alf7e3f2ec4cef8ecb9c67a65bff3c18acc1109d9e9ab5397f326010
timestamp: 2023-05-09 18:25:59.018541
2 block created
Hash: 2f54e5cccd292057f5aaeb14d28054763fd91d34bb69dc50805c56ca65dc6ab4
timestamp: 2023-05-09 18:25:59.049793
```

Practical No 2

2a) Aim: Create a smart contract for counter.

Code:

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract Counter {
6
7     uint256 private count=0;
8
9     function getCount() public view returns (uint256) {
10         return count;
11     }
12
13     function increment() public { infinite gas
14         count += 1;
15     }
16 }
```

Output:

Deployed Contracts

COUNTER AT 0X7EF...8CB47 (MEMORY)

Balance: 0 ETH

increment

getCount

0: uint256: 3

2b) Aim: Create a smart contract for calculator.

Code:

```
2_b_calculator.sol X
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity >=0.8.2 <0.9.0;
4
5  contract Calculator {
6
7      int256 number1;
8      int256 number2;
9
10     function inputNumbers(int256 num1, int256 num2) public {  infinite gas
11         number1 = num1;
12         number2 = num2;
13     }
14
15     function add() public view returns (int256){  infinite gas
16         int256 result = number1 + number2;
17         return result;
18     }
19
20     function sub() public view returns (int256){  infinite gas
21         int256 result = number1 - number2;
22         return result;
23     }
24
25     function mul() public view returns (int256){  infinite gas
26         int256 result = number1 * number2;
27         return result;
28     }
29
30     function div() public view returns (int256){  infinite gas
31         int256 result = number1 / number2;
32         return result;
33     }
34 }
35 }
```

Output:

Deployed Contracts

▼ CALCULATOR AT 0XDDA...5482D (MEMORY)

Balance: 0 ETH

inputNumbers

num1:

num2:

Calldata

Parameters

transact

add

0: int256: 6

div

0: int256: 2

mul

0: int256: 8

sub

0: int256: 2

Practical No 3



3a) Aim: Write a solidity program to demonstrate array and its types.

Code:

```
3_a_arrays.sol x
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract ArrayTypes {
6
7     //< ===== Fixed Size Array =====>
8     uint[4] rollNumbers;
9
10    function setRollNumbers(uint _index, uint _rollNumber) public { infinite gas
11    |
12    |     rollNumbers[_index] = _rollNumber;
13    |
14    | }
15
16    function getRollNumbers() public view returns(uint[4] memory) { infinite gas
17    |
18    |     return rollNumbers;
19    |
20    | }
21
22    //< ===== Dynamic Size Array =====>
23    uint[] employeeIds;
24
25    function setEmployeeIds(uint _empId) public { 46873 gas
26    |
27    |     employeeIds.push(_empId);
28    |
29    | }
30
31    function removeEmployeeId() public { 29511 gas
32    |
33    |     employeeIds.pop();
34    |
35    | }
36
37    function getEmployeeIds() public view returns(uint[] memory) { infinite gas
38    |
39    |     return employeeIds;
40    |
41    | }
42
43    //< ===== Multi-Dimensional Array =====>
44
45    // uint256[col][row] array_name
46    uint[2][3] public data = [[1,2], [3,4],[5,6]];
47
48    function getData(uint _col, uint _row) public view returns (uint) { infin
49    |
50    |     uint number = data[_col][_row];
51    |     return number;
52    |
53    | }
54 }
```

Output:

Deployed Contracts

▼ ARRAYTYPES AT 0XDDA...5482D (MEM)  

Balance: 0 ETH

removeEmploy

setEmployeeId

30

▼

setRollNumber

3,5

▼

data

0,0

▼

0: uint256: 1

getData

0,0

▼

0: uint256: 1

getEmployeeId

0: uint256[]: 10,20

getRollNumber

0: uint256[4]: 2,3,4,5

3b) Aim: Demonstrate the use of loops in solidity.

Code:

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity ^0.8.0;
4
5 contract Loops {
6
7     // while loop
8     uint256 private result=0;
9     function whileLoop(uint256 number) public returns (uint256) { infinite gas
10         uint256 i = 1;
11         while (i <= number) {
12             result += i;
13             i++;
14         }
15         return result;
16     }
17
18     function getWhileLoop() public view returns (uint256) { 2503 gas
19         return result;
20     }
21
22     // do while
23     uint256 private result1=0;
24     function doWhileLoop(uint256 number) public returns(uint256) { infinite gas
25         uint256 i = 1;
26         do {
27             result1 += i;
28             i++;
29         } while(i <= number);
30         return result1;
31     }
32
33     function getDoWhileLoopResult() public view returns (uint256) { 2437 gas
34         return result1;
35     }
36
37
38     // for loop
39     uint256 private result2=0;
40
41     function forLoop(uint256 number) public returns(uint256) { infinite gas
42         for(uint256 i=1;i<=number;i++) {
43             result2 +=i;
44         }
45         return result2;
46     }
47
48     function geForLoop() public view returns (uint256) { 2415 gas
49         return result2;
50     }
51
52 }
```

Output:

Deployed Contracts

▼

LOOPS AT 0X332...D4B6D (MEMORY)

×

Balance: 0 ETH

doWhileLoop

6

▼

forLoop

7

▼

whileLoop

5

▼

geForLoop

0: uint256: 28

getDoWhileLo

0: uint256: 21

getWhileLoop

0: uint256: 15

Practical No 4

Aim: Write a solidity program to demonstrate the use of following operators:

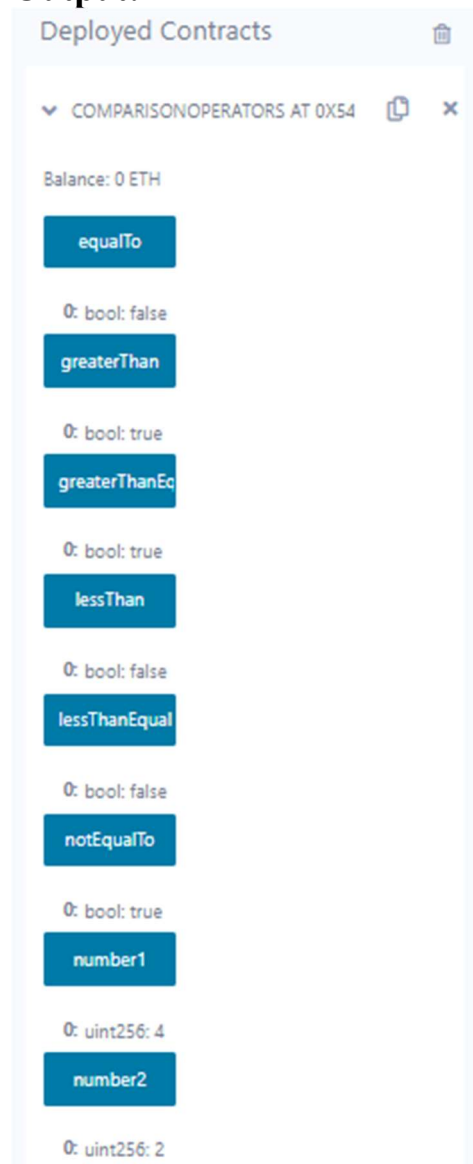
a) Comparison b) Logical c) Assignment d) Ternary e) bitwise

4a) Aim: Solidity program to demonstrate Comparison operators.

Code:

```
4_b_LogicalOperators.sol 2 4_a_ComparisonOperators.sol X
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract ComparisonOperators {
6
7     uint public number1 = 4;
8     uint public number2 = 2;
9
10    // ==
11    function equalTo() public view returns (bool) { 4584 gas
12        bool result = number1 == number2;
13        return result;
14    }
15
16    // !=
17    function notEqualTo() public view returns (bool) { 4610 gas
18        bool result = number1 != number2;
19        return result;
20    }
21
22    // <
23    function lessThan() public view returns (bool) { 4563 gas
24        bool result = number1 < number2;
25        return result;
26    }
27
28    // <=
29    function lessThanEqual() public view returns (bool) { 4609 gas
30        bool result = number1 <= number2;
31        return result;
32    }
33
34    // >
35    function greaterThan() public view returns (bool) { 4628 gas
36        bool result = number1 > number2;
37        return result;
38    }
39
40    // >=
41    function greaterThanEqual() public view returns (bool) { 4632 gas
42        bool result = number1 >= number2;
43        return result;
44    }
45 }
```

Output:



4b) Aim: Solidity program to demonstrate Logical operators.

Code:

```
4_b_LogicalOperators.sol X
1  // SPDX-License-Identifier: GPL-3.0
2
3  pragma solidity >=0.8.2 <0.9.0;
4
5  contract ComparisonOperators {
6
7      bool public amazonLogin = true;
8      bool public netflixLogin = false;
9
10     // !
11     function logicalNot() public view returns (string memory) { infinite gas
12         string memory result;
13         if(!netflixLogin) {
14             result = "user is not logged in to netflix.";
15         } else {
16             result = "user is logged in to netflix.";
17         }
18         return result;
19     }
20
21     // &&
22     function logicalAND() public view returns (string memory) { infinite gas
23         string memory result;
24         if(netflixLogin && amazonLogin) {
25             result = "User has logged in to both websites.";
26         } else {
27             result = "User has not logged in to one of the website.";
28         }
29         return result;
30     }
31
32     // ||
33     function logicalOR() public view returns (string memory) { infinite gas
34         string memory result;
35         if(netflixLogin || amazonLogin) {
36             result = "User is logged in to one of the websites.";
37         } else {
38             result = "User is not logged in to both of the websites.";
39         }
40         return result;
41     }
42 }
```

Output:

The screenshot displays a web application interface for managing deployed smart contracts. At the top, a header bar contains the title 'Deployed Contracts' and a trash icon. Below this, a dropdown menu is open, showing a list of contracts. The selected contract is 'COMPARISONOPERATORS AT 0X38C..', which has a copy icon and a close icon. The main content area shows the details of the selected contract. It starts with the text 'Balance: 0 ETH'. Below this, there is a button labeled 'amazonLogin'. To the right of this button is a grey callout box containing the text 'amazonLogin - call'. Below the button, the output of the function call is displayed as '0: bool: true'. This is followed by a button labeled 'logicalAND'. Below this button, the output is '0: string: User has not logged in to one of the website.'. This is followed by a button labeled 'logicalNot'. Below this button, the output is '0: string: user is not logged in to netflix.'. This is followed by a button labeled 'logicalIOR'. Below this button, the output is '0: string: User is logged in to one of the websites.'. This is followed by a button labeled 'netflixLogin'. Below this button, the output is '0: bool: false'.

Deployed Contracts

▼ COMPARISONOPERATORS AT 0X38C..

Balance: 0 ETH

amazonLogin

amazonLogin - call

0: bool: true

logicalAND

0: string: User has not logged in to one of the website.

logicalNot

0: string: user is not logged in to netflix.

logicalIOR

0: string: User is logged in to one of the websites.

netflixLogin

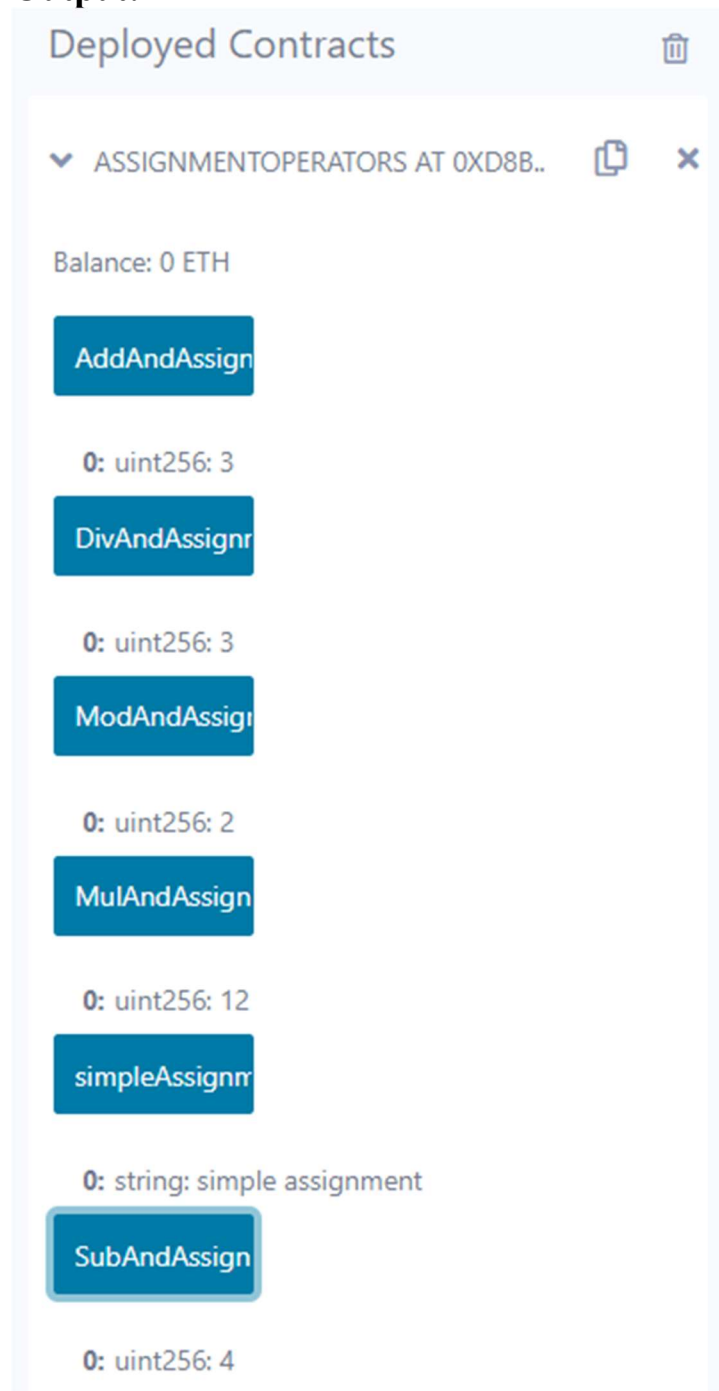
0: bool: false

4c) Aim: Solidity program to demonstrate Assignment operators.

Code:


```
4_c_AssignmentOperator.sol X
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract AssignmentOperators {
6
7     // =
8     function simpleAssignment() public pure returns (string memory) { infinite gas
9         string memory result = "simple assignment";
10        return result;
11    }
12
13    // +=
14    function AddAndAssignment() public pure returns (uint) { infinite gas
15        uint result = 1;
16        result +=2;
17        return result;
18    }
19
20    // -=
21    function SubAndAssignment() public pure returns (uint) { infinite gas
22        uint result = 6;
23        result -=2;
24        return result;
25    }
26
27    // *=
28    function MulAndAssignment() public pure returns (uint) { infinite gas
29        uint result = 6;
30        result *=2;
31        return result;
32    }
33
34    // /=
35    function DivAndAssignment() public pure returns (uint) { infinite gas
36        uint result = 6;
37        result /=2;
38        return result;
39    }
40
41    // %=
42    function ModAndAssignment() public pure returns (uint) { infinite gas
43        uint result = 6;
44        result %=4;
45        return result;
46    }
47 }
```

Output:






4d) Aim: Solidity program to demonstrate Ternary operators.

Code:


```
4_c_AssignmentOperator.sol 4_d_ternary.sol X
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract TernaryOperators {
6
7     bool public loggedIn = true;
8
9     function ternaryOperator() public view returns (string memory) {  infinite gas
10         string memory result = (loggedIn) ? "user is loggedIn" : "user is not loggedIn";
11         return result;
12     }
13 }
```

Output:

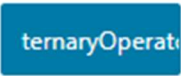
Deployed Contracts 

▼ TERNARYOPERATORS AT 0XF8E...9FBI  

Balance: 0 ETH



0: bool: true



0: string: user is loggedIn

4e) Aim: Solidity program to demonstrate Bitwise operators.

Code:

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract TernaryOperators {
6
7     int public a = 10;
8     int public b = 12;
9
10    // & (Bitwise AND)
11    int public OfAND = a & b;
12
13    // | (Bitwise OR)
14    int public OfOR = a | b;
15
16    // ^ (Bitwise XOR)
17    int public OfXOR = a ^ b;
18
19    // ~ (Bitwise Not)
20    int public OfNOT = (~a);
21
22    // << (Left Shift)
23    int public OfLEFTSHIFT = a << 2;
24
25    // >> (Right Shift)
26    int public OfRIGHTSHIFT = a >> 2;
27
28 }
```

Output:

Deployed Contracts

TERNARYOPERATORS AT 0XD2A...FD005 (M)

Balance: 0 ETH

a
0: int256: 10
b
0: int256: 12
OfAND
0: int256: 8
OfLEFTSHIFT
0: int256: 40
OfNOT
0: int256: -11
OfOR
0: int256: 14
OfRIGHTSHIFT
0: int256: 2
OfXOR
0: int256: 6

Practical No 5

Aim: Write a solidity program to find if the number is odd or even, prime or composite.

Code:

```
4_e_bitwise.sol 5_evenOddPrimeComposite.sol X
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract evenOddPrimeComposite {
6
7     uint public number;
8
9     function setNumber(uint _number) public { 22498 gas
10         number = _number;
11     }
12
13     function primeCompositeEvenOddChecker () public view returns(string memory, string memory, string memory) {
14
15         string memory resultOfPrime;
16         string memory resultOfComposite;
17         string memory resultOfEvenOdd;
18
19         if(number <=1) {
20             resultOfPrime = "not a prime number.";
21             resultOfComposite = "not a composite number.";
22         }
23         for(uint i =2;i<number/2;i++) {
24             if(number % i == 0) {
25                 resultOfPrime = "number is not prime.";
26                 resultOfComposite = "number is composite.";
27             }
28         }
29
30         if(number %2 == 0) {
31             resultOfEvenOdd = "number is even.";
32         } else {
33             resultOfEvenOdd = "number is odd.";
34         }
35         return (resultOfPrime, resultOfComposite, resultOfEvenOdd);
36     }
37 }
38
```

Output:

Deployed Contracts

▼ EVENODDPRIMECOMPOSITE AT 0XB27...07

Balance: 0 ETH

setNumber

6

▼

number

0: uint256: 6

primeComposi

0: string: number is not prime.
1: string: number is composite.
2: string: number is even.

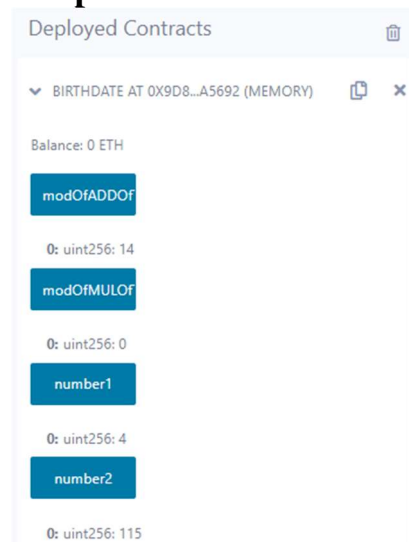
Practical No 6

Aim: Write a solidity program to find modulus of addition of two numbers with DD and modulus of multiplication of two numbers with MM where the two numbers are achieved by performing AND operation of DD and MM of your date of birth and OR operation of YY and YY from year of birth.

Code:

```
2
3  pragma solidity >=0.8.2 <0.9.0;
4
5  contract Birthdate {
6      uint DD = 21;
7      uint MM = 4;
8      uint YY = 19;
9      uint YYM = 99;
10
11     uint public number1 = DD & MM;
12     uint public number2 = YY | YYM;
13
14     uint public modOfADDOfTwoNumbers = addmod(number1, number2, DD);
15     uint public modOfMULOOfTwoNumbers = addmod(number1, number2, MM);
16 }
17
```

Output:



Practical No 7

7a) Aim: Create a smart contract to demonstrate Mathematical function.

Code:

```
7_a_mathematicalFunctions.sol X
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract MathematicalFunction {
6
7     uint public number1 = 7;
8     uint public number2 = 8;
9
10    uint public modOfADDOfTwoNumbers = addmod(number1, number2, 2);
11    uint public modOfMULOOfTwoNumbers = mulmod(number1, number2, 4);
12 }
13
```

Output:

Deployed Contracts

MATHEMATICALFUNCTION AT 0XD4F...2CE

Balance: 0 ETH

modOfADDOf

0: uint256: 1

modOfMULOOf

0: uint256: 0

number1

0: uint256: 7

number2

0: uint256: 8

7b) Aim: Create a smart contract to demonstrate Function overloading.

Code:

```
7_b_functionOverloading.sol
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract functionOverloading {
6
7     function sum(uint _a, uint _b) public pure returns(uint) {
8         uint result = _a + _b;
9         return result;
10    }
11
12    function sum(uint _a, uint _b, uint _c) public pure returns(uint) {
13        uint result = _a + _b + _c;
14        return result;
15    }
16
17    function caller() public pure returns(uint, uint) {
18        return (sum(2,3), sum(5,6,7));
19    }
20 }
```

Output:

Deployed Contracts

FUNCTIONOVERLOADING AT 0X332...D486

Balance: 0 ETH

caller

0: uint256: 5
1: uint256: 18

sum uint256 _a, uint256 _b, uint256 _c

sum uint256 _a, uint256 _b

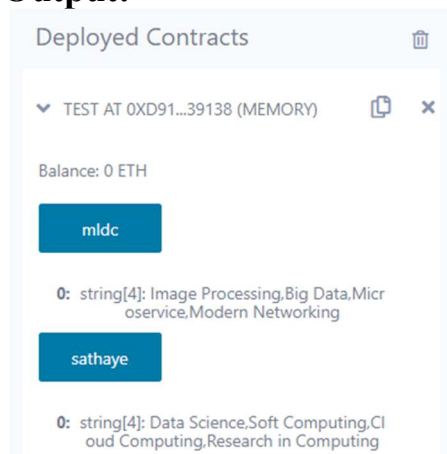
Practical 8

8a) Aim: Create a smart contract to show the implementation of Interface.

Code:

```
8_a_interfaceImplementation.sol X
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 /* Consider university of mumbai as interface where as to provide subject electives to two colleges sathaye
6 and mldc(methods inside interface),
7 create a smart contract to provide the list of sem-1 subjects in the form of array to sathaye and sem-2 subjects to mldc.
8 */
9
10
11 interface MumbaiUniversity {
12     function sathaye() external pure returns (string[4] memory);
13     function mldc() external pure returns (string[4] memory);
14 }
15
16 contract Test is MumbaiUniversity {
17
18     function sathaye() public pure override returns (string[4] memory) {
19         string[4] memory sem_1 = ["Data Science", "Soft Computing", "Cloud Computing", "Research in Computing"];
20         return sem_1;
21     }
22
23     function mldc() public pure override returns (string[4] memory) {
24         string[4] memory sem_2 = ["Image Processing", "Big Data", "Microservice", "Modern Networking"];
25         return sem_2;
26     }
27 }
28
```

Output:



8b) Aim: Create a smart contract to show the implementation of Inheritance.

Code:

```
8_b_inheritanceImplementation.sol X
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 /*
6 Consider university of mumbai as the base class allocating array of roll numbers to two colleges
7 mldc and sathaye(methods in class).
8 Where enrollment will happen and hence create a smart contract to place the first ten roll numbers
9 in mldc contract and last ten roll numbers in sathaye contract, also find the factorial of first four numbers in
10 each contract.
11 */
12
13 contract MumbaiUniversity {
14     uint256[] internal rollNumbers;
15
16     constructor() {
17         for (uint256 i = 0; i < 20; i++) {
18             rollNumbers.push(i + 3);
19         }
20     }
21 }
22
23 contract MLDC is MumbaiUniversity {
24     uint256[] private enrolledToMLDC;
25
26     constructor() {
27         for (uint256 i = 0; i < 10; i++) {
28             enrolledToMLDC.push(rollNumbers[i]);
29         }
30     }
31
32     function getEnrolledM() public view returns (uint256[] memory) {
33         return enrolledToMLDC;
34     }
35
36     function factorialOfFirstFour() public view returns (uint256[4] memory) {
37         uint256[4] memory factorials;
38         for (uint256 i = 0; i < 4; i++) {
39             uint256 num = enrolledToMLDC[i];
40             uint256 factorial = 1;
41             for (uint256 j = 2; j <= num; j++) {
42                 factorial *= j;
43             }
44             factorials[i] = factorial;
45         }
46         return factorials;
47     }
48 }
49
50
```

```
51 contract Sathaye is MumbaiUniversity {
52     uint256[] private enrolledToSATHAYE;
53
54     constructor() {  infinite gas 219000 gas
55         for (uint256 i = 10; i < rollNumbers.length; i++) {
56             enrolledToSATHAYE.push(rollNumbers[i]);
57         }
58     }
59
60     function getEnrolledS() public view returns (uint256[] memory) {  infinite gas
61         return enrolledToSATHAYE;
62     }
63
64     function factorialOfFirstFour() public view returns (uint256[4] memory) {  infinite gas
65
66         uint256[4] memory factorials;
67         for (uint256 i = 0; i < 4; i++) {
68             uint256 num = enrolledToSATHAYE[i];
69             uint256 factorial = 1;
70             for (uint256 j = 2; j <= num; j++) {
71                 factorial *= j;
72             }
73             factorials[i] = factorial;
74         }
75         return factorials;
76     }
77 }
```

Output:

Deployed Contracts

▼ MUMBAIUNIVERSITY AT 0X540...C75E

Balance: 0 ETH

0: uint256[]: 3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22

Deployed Contracts

▼ MLDC AT 0XEF9...10EBF (MEMORY)

Balance: 0 ETH

0: uint256[4]: 6,24,120,720

0: uint256[]: 3,4,5,6,7,8,9,10,11,12

0: uint256[]: 3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22

Deployed Contracts

▼ SATHAYE AT 0X38C...24C73 (MEMORY)

Balance: 0 ETH

0: uint256[4]: 6227020800,87178291200,1307674368000,20922789888000

0: uint256[]: 13,14,15,16,17,18,19,20,21,22

0: uint256[]: 3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22

Practical No 9

Aim: Write a solidity program to create an array of roll numbers and then create a smart contract where it checks the value of the roll number and perform AND operation with today's date and if the result is even then allow the student else deny (DD part only).

Code:

```
9_allowDenyStudent.sol X
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract allowDenyStudent {
6     uint256[] public arr = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10];
7     uint256 public rollNumber;
8     uint256 public DD;
9
10    function setRollNumber(uint256 _rollNumber) public { 26800 gas
11        rollNumber = arr[_rollNumber];
12    }
13
14    function setDD(uint256 _DD) public { 22498 gas
15        DD = _DD;
16    }
17
18    function arrayChecker() public view returns (string memory) {
19        string memory finalOutput;
20
21        uint256 operation = rollNumber & DD;
22
23        if (operation % 2 == 0) {
24            finalOutput = "student is allowed.";
25        } else {
26            finalOutput = "student is denied.";
27        }
28
29        return finalOutput;
30    }
31 }
```

Output:

Deployed Contracts

▼ ALLOWDENYSTUDENT AT 0XDA0...5D3DF (M)

Balance: 0 ETH

setDD

9

▼

setRollNumber

3

▼

arr

uint256

▼

arrayChecker

0: string: student is allowed.

DD

0: uint256: 9

rollNumber

0: uint256: 4

Practical No 10

Aim: Write a solidity program to find the sum of an array of ten numbers using loop the numbers are expected to be taken from the user, create a smart contract to find the AND operation of odd positioned numbers and OR operation of even positioned numbers including 0th index. Hence find the product of the results and also identify whether the result is the part of array or not.

Code:

```
1 // SPDX-License-Identifier: GPL-3.0
2
3 pragma solidity >=0.8.2 <0.9.0;
4
5 contract allowDenyStudent {
6
7     uint256[10] public rollNumbers;
8
9     function setRollNumber(uint _index ,uint _rollNumber) public {
10         // push() method is not available on fixed size array.
11         rollNumbers[_index] = (_rollNumber);
12     }
13
14     function seperatingEvenOdd() public {
15
16         for(uint i=0;i<rollNumbers.length;i++) {
17
18             if(i %2 == 0) {
19                 evenPositioned.push(rollNumbers[i]);
20             } else {
21                 oddPositioned.push(rollNumbers[i]);
22             }
23         }
24     }
25
26     uint[] public evenPositioned;
27     uint[] public oddPositioned;
28
29     function OR_AND_Operation() public {
30
31         uint resultOfOR = evenPositioned[0];
32         for(uint i=1;i<evenPositioned.length;i++) {
33             resultOfOR = resultOfOR | evenPositioned[i];
34         }
35
36         uint resultOfAND = oddPositioned[0];
37         for(uint i=1;i<oddPositioned.length;i++) {
38             resultOfAND = resultOfAND & oddPositioned[i];
39         }
40         productOfresults = resultOfOR * resultOfAND;
41     }
42
43     uint256 public productOfresults;
44
45     function checkProductOfResults() public view returns(bool) {
46         bool result;
47         for(uint i=0; i<rollNumbers.length; i++) {
48
49             result = (productOfresults == rollNumbers[i]) ? true : false;
50         }
51         return result;
52     }
53 }
```

Output:

Deployed Contracts

ALLOWDENYSTUDENT AT 0XD91...

Balance: 0 ETH

OR_AND_Open

seperatingEven

setRollNumber

_index: "9"

_rollNumber: "10"

CalldataParameterstransact

checkProductOfresu

0: bool: false

evenPositioned 0

0: uint256: 1

oddPositioned 0

0: uint256: 2

productOfresu

0: uint256: 0

rollNumbers 9

0: uint256: 10